Downloaded from orbit.dtu.dk on: May 04, 2023

DTU Library

A Time-predictable Memory Network-on-Chip

Schoeberl, Martin; Chong, David VH; Puffitsch, Wolfgang; Sparsø, Jens

Published in:
Proceedings of the 14th International Workshop on Worst-Case Execution Time Analysis (WCET 2014)

Link to article, DOI:
10.4230/OASIcs.WCET.2014.53

Publication date:
2014

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):
Schoeberl, M., Chong, D. VH., Puffitsch, W., & Sparsø, J. (2014). A Time-predictable Memory Network-on-Chip. In H. Falk (Ed.), *Proceedings of the 14th International Workshop on Worst-Case Execution Time Analysis (WCET 2014)* (pp. 53-62). OASICS. Open Access Series in Informatics Vol. 39
https://doi.org/10.4230/OASIcs.WCET.2014.53

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

A Time-Predictable Memory Network-on-Chip

Martin Schoeberl, David Vh Chong, Wolfgang Puffitsch, and Jens Sparsø

Department of Applied Mathematics and Computer Science
Technical University of Denmark
masca@dtu.dk, davidchong99@gmail.com, wopu@dtu.dk, jspa@dtu.dk

Abstract

To derive safe bounds on worst-case execution times (WCETs), all components of a computer system need to be time-predictable: the processor pipeline, the caches, the memory controller, and memory arbitration on a multicore processor. This paper presents a solution for time-predictable memory arbitration and access for chip-multiprocessors. The memory network-on-chip is organized as a tree with time-division multiplexing (TDM) of accesses to the shared memory. The TDM based arbitration completely decouples processor cores and allows WCET analysis of the memory accesses on individual cores without considering the tasks on the other cores. Furthermore, we perform local, distributed arbitration according to the global TDM schedule. This solution avoids a central arbiter and scales to a large number of processors.

1998 ACM Subject Classification B.4.3 Interconnections (Subsystems)

Keywords and phrases Real-Time Systems, Time-predictable Computer Architecture, Network-on-Chip, Memory Arbitration

Digital Object Identifier 10.4230/OASIcs.WCET.2014.53

1 Introduction

The trend in processor design is to increase performance by including more and more processing cores in a single chip. This has been accompanied by a shift from bus-based interconnects to some form of packet switched networks-on-chip (NoC), because chip-wide single-cycle communication has become infeasible. Typically the on-chip processors share an external memory for large shared data structures and for program code. A dedicated NoC is often used for this communication. The NoC and the shared memory are shared resources and in general purpose processors they are a source of timing interferences between tasks executing on different processor cores.

To enable static worst-case execution time (WCET) analysis of applications executing on a multicore processor, both the individual processor cores and the shared memory system (including the NoC) need to be time-predictable [19]. This paper presents the design for timing predictability of the memory interconnect for a chip-multiprocessor. The presented memory system offers time-composability where the execution times of different tasks executing on different processor cores are independent of each other.

Figure 1 shows the multicore platform, as it is developed in the T-CREST project. Several processor cores, the Patmos processors [22], are connected to two NoCs: (1) a core NoC for message passing between processor-local scratchpad memories [7, 21, 24], and (2) a memory NoC – the focus of this paper – that connects all processor cores to the shared, external memory via the memory controller.

The main idea of the presented design is to use TDM scheduling from end to end, such that read or write transactions towards the shared memory are transmitted from the


© Martin Schoeberl, David Vh Chong, Wolfgang Puffitsch, and Jens Sparsø;
licensed under Creative Commons License CC-BY
14th International Workshop on Worst-Case Execution Time Analysis (WCET 2014).
Editor: Heiko Falk; pp. 53–62

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

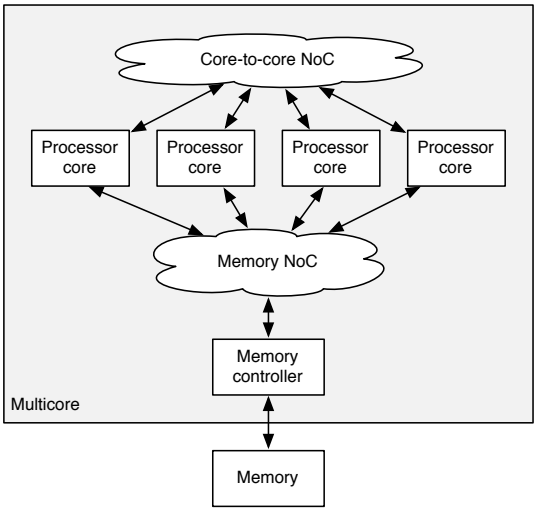

Figure 1 Multicore architecture with several processor cores connected to two NoCs: one for core-to-core message passing and one for access to the shared, external memory.

initiating processor core to the memory without any dynamic arbitration or buffering. Only the processor-local memories (caches and/or scratchpad-memories) buffer any data. By injecting transactions according to a global schedule, they can be propagated in a pipelined fashion even without flow control. The TDM slots and the TDM schedule are defined by the sequence of equally sized read or write transactions towards the memory.

Compared to implementations that use source rate control and dynamic arbitration, the use of TDM results in both a simple hardware implementation and a straightforward WCET analysis. Furthermore, executing this global TDM schedule distributed at the processor cores results in distributed arbitration that scales well with increased number of processor cores. Compared to other TDM memory arbiters we consider pipelining in the design and account for the pipeline delays in the timing parameters. This is the contribution of the paper.

The paper is organized in 6 sections: Section 2 presents related work. Section 3 presents the design of the memory NoC and Section 4 the resulting timing of memory transactions and TDM slotting. Section 5 presents the implementation and evaluation of the memory NoC in an FPGA. Section 6 concludes.

2 Related Work

The design of time-predictable multicore systems is attracting increasing interest. Cullman et al. discusses some high-level design guidelines [3]. To simplify WCET analysis (or even make it feasible) the architecture shall be *timing compositional*. That means that the architecture has no timing anomalies or unbounded timing effects [12]. The Patmos processor, used in the proposed time-predictable multicore, fulfills those properties. For multicore systems the authors of [3] argue for bounded access delays on shared resources. This is in our opinion best fulfilled by a TDM based arbitration scheme, as presented in this paper.

From a structural point of view, communication between processor cores and external memory is different from inter-core communication. While the former follows a many-to-one communication pattern, the latter requires many-to-many communication. Consequently, approaches to make many-to-many communication predictable [5, 13, 24] are not directly comparable to the work presented in this paper.

The proposed memory NoC is similar to "mesh-of-trees" NoCs [2, 17] with a single tree. These NoCs perform 2:1 arbitration in the nodes of the trees such that a request that is blocked by a request on the other input will win arbitration in the next cycle. Therefore, arbitration follows a distributed round-robin scheme.

Different arbitration schemes for a time-predictable memory access of a processor and a video controller are evaluated in [15]. The work has been extended to build a time-predictable multicore version of the Java processor JOP [16]. A TDM based memory arbitration is used and the WCET of individual bytecodes of the Java processor take the possible positions within the TDM schedule into account. Therefore, some latency introduced by TDM arbitration can be hidden. In contrast to our distributed TDM memory arbiter, the JOP TDM arbiter was designed to include the read response within the TDM slot. This design limits the possibility to pipeline the arbiter.

The initial memory connection in the T-CREST project is the so-called Bluetree [6]. Bluetree is a tree of 2:1 multiplexers where the default behavior is that one of the inputs has priority over the other. To avoid starvation of the lower priority input, a counter controls the maximum number of priority messages when a low priority message is pending. This design is optimized to deliver good average-case performance and guarantee worst-case responses. Compared to the Bluetree, our design is not work conserving, but has a shorter worst-case latency guarantee, considering all other parameters the same.

An approach close to our work is presented in [18]. The proposed multicore system is also intended for tasks according to the simple task model [9]. The local cache loading for the processor cores is performed from a shared main memory. Similar to our approach, a TDM based memory arbitration is used. The memory wheel of the PRET architecture [11] is also a form of TDM arbitration. PRET's memory wheel arbitrates between the six hardware threads of processor rather than between different processor cores. Therefore, scalability is not a major concern.

Gomony at al. argue for coupling the NoC TDM slotting with the TDM arbitration within a memory controller [4]. This is in line with our argument. However, compared to their design we completely avoid buffering in the NoC and in the memory controller.

Schranzhofer et al. [23] present a timing analysis for TDM arbitration. They model the case where memory accesses are allowed only at the beginning and end of a task, the case where accesses are allowed at any time, and a hybrid of these two cases. Kelter et al. [8] also present a timing analysis for TDM arbitration. They compare several variants that trade off precision and analysis effort. They find that their approach can lead to significantly lower WCETs for timing-composable systems than for non-composable systems.

Most TDM based designs consider equal slots for all processor cores. However, it is easy to envision a schedule where some cores get more than one slot per TDM period. One can optimize this allocation with the WCET of the individual tasks running on the different processor cores. However, it has been shown that this rather coarse grain optimization is not very efficient [25].

3 Memory Network-on-Chip Design

Figure 2 shows the memory NoC design. As this NoC serves a many-to-one communication flow between several processor cores and a single memory controller, it is organized as a tree. Each processor core is connected via a standard interface, the open core protocol (OCP) [14], to the network interface (NI). The NIs are connected by a tree of merge circuits downstream towards the memory interface (MI) and back upstream for the return data. The

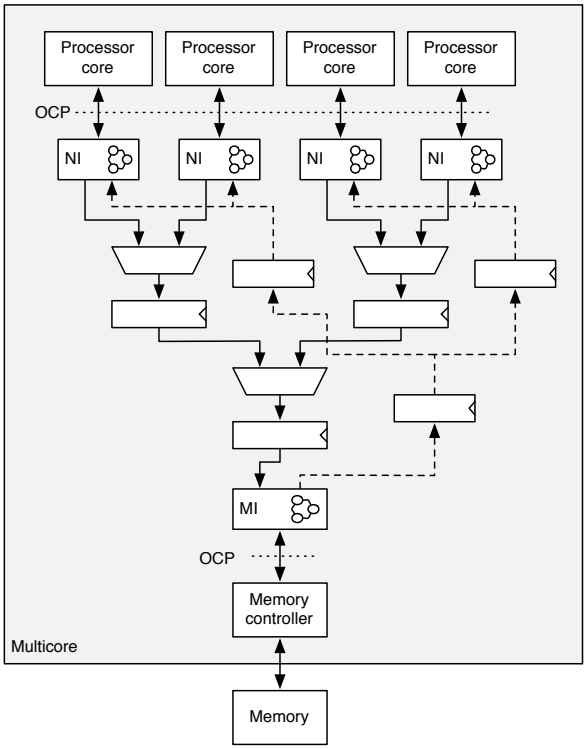

Figure 2 The distributed TDM based memory NoC.

MI is connected to the on-chip memory controller, which itself is connected to the external memory.

The memory NoC supports burst read and write transactions. For single word/byte writes, write enable signals for individual bytes are supported. When the external memory is a DRAM device that needs refresh, a refresh circuit is added to the memory NoC at the same level as a processor core. Therefore, refresh consumes one TDM slot, but has no further influence on the memory access timing.

Each core local NI executes the core relevant part of the global TDM schedule. When the time slot for a core arrives and a memory transaction is pending, the NI acknowledges the transaction to the processor core and the transaction data freely flows down the network tree. No flow control, arbitration, or buffering (except pipeline registers to improve clock frequency) is performed along the downstream path. The memory request arrives at the MI and is translated back to an OCP transaction request to the memory controller. Here OCP handshaking is generated, but the TDM schedule is organized such that it is guaranteed that the memory controller and the memory are ready to accept the transaction.

On a read transaction, the result is returned from the memory to the memory controller and from there back upstream to the processor cores. The returning to the processor cores can be a simple broadcast to all processors, which itself can be pipelined. Alternatively, it can be organized as a broadcast tree as shown in Figure 2. Due to the pipelining, several read requests might be on the fly in the memory NoC, memory controller, and memory. Therefore, a processor might see a read return from a former read request by a different processor after sending the read request.

To identify the correct return data there are two possibilities: (1) either tag the memory transaction with the core number or (2) use time to distinguish between the early and false

Table 1 Timing parameters for the memory interface, the memory controller, and the memory NoC.

Parameter	Meaning
t_b	Burst transfer length
t_{r2b}	Read command to read burst delay
t_{b2e}	Write burst to command end delay
$t_{rd} = t_{r2b} + t_b$	Read transaction timing (at the memory interface)
$t_{wr} = t_b + t_{b2e}$	Write transaction timing (at the memory interface)
t_{ctrlrd}, t_{ctrlwr}	Non-pipelinable time delays in the memory controller
$t_{slot} = \max(t_{rd} + t_{ctrlrd}, t_{wr} + t_{ctrlwr})$	Minimum slot length
N	Number of processor cores
$T = N \times t_{slot}$	TDM period (with equal memory bandwidth)
L_{down}	Memory NoC and controller downstream latency
L_{up}	Memory NoC and controller upstream latency
$t_{wcrd} = T - 1 + L_{down} + t_{slot} + L_{up}$	Worst-case read transaction time
$t_{wcwr} = T - 1 + t_{slot}$	Worst-case write transaction time
$t_{bcrd} = L_{down} + t_{slot} + L_{up}$	Best-case read transaction time
$t_{bcwr} = t_{slot}$	Best-case write transaction time

and the correct read responses. With a tagged transaction and pipelining in the memory controller and memory, the memory controller needs to organize a short queue of the tags of outstanding read requests. As a positive side effect, this information can be used to send the return information only to the originally requesting processor core along the upstream tree, saving power in the other paths. Using time to distinguish between read returns is possible with the static TDM schedule and simple to implement in the local NI. The NI knows the latency for the read request and simply ignores any receiving data until this latency has passed.

For a write transaction there are two options for the handshake between the processor core and the memory NoC: (1) just post the write data and generate the write acknowledgement locally or (2) wait for the write and the write acknowledgment from the memory controller. Just posting the write lets the processor pipeline continue to execute code and hides the memory NoC latency. As the memory NoC does not reorder any memory transactions there is no issue with memory consistency. If the memory and/or controller supports error return codes (e.g., on a parity error) waiting for the response enables error signaling with an exception in the processor core.

The design is configurable with 4 parameters: (1) the level of downstream pipelining, (2) the maximum time for a memory transaction (i.e, the TDM slot length), (3) the additional latency within the memory controller, and (4) the level of upstream pipelining. The pipeline levels influence downstream and upstream latencies.

4 Pipelined Access Timing and Slot Length

We are mainly interested in the worst-case access time for a memory transaction (cache miss). For some scenarios, it might also be interesting what the best-case access time is. For WCET analysis, where we look at a sequence of memory accesses with operations in between, we are interested in how much of the access latency can be overlapped with execution and therefore hidden.

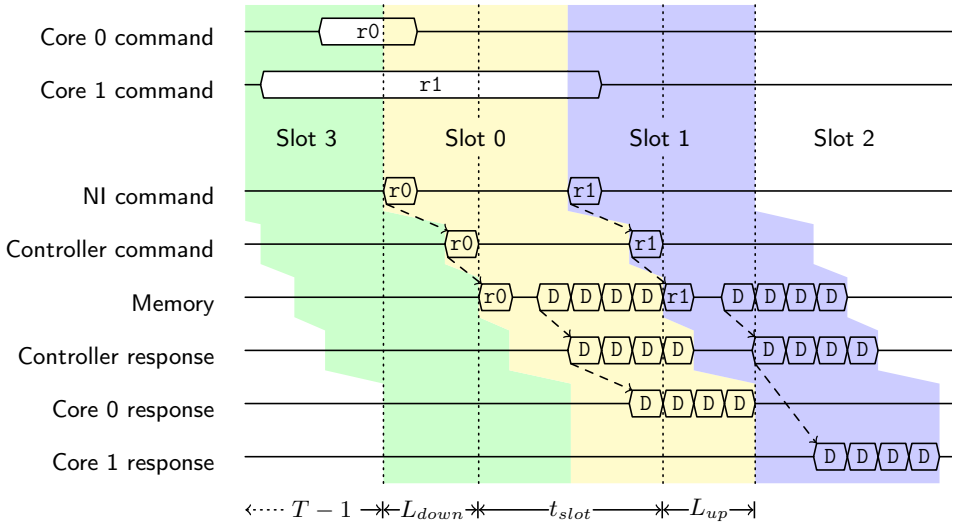

Figure 3 Access latency, TDM slot length, and slot shifting due to pipelining.

Table 1 shows all timing parameters, which are in clock cycles. Timing parameters with a lower case t in the name denote non-pipeline delays and parameters with an upper case L delays that can be pipelined. E.g., the bursting of data via the pins of the memory chips cannot be pipelined, but the burst of data traveling in the memory NoC can be pipelined.

Memory read and write transactions at the memory chip take t_b clock cycles for the burst transfer of the data and some additional latency for a read command to process (t_{r2b}) and a write command to finish (t_{b2e}). Due to some inefficiency in the memory controller, additional delays on read or write transaction (t_{ctrlrd} and t_{ctrlwr}) may be introduced that cannot be hidden by pipelining. The combination of those memory and controller delays determines the minimum TDM slot length t_{slot}.

The slot length t_{slot} and the number of processing cores determine the TDM period T.[1] The memory NoC (and the memory controller) introduce additional latency that needs to be added for the worst-case access time (t_{wcrd} and t_{wcwr}). However, even with a low number of processing cores the TDM period T is the main contributor to the access latency.

Figure 3 shows two read transactions by cores 0 and 1 in a configuration with four cores. The accesses of the cores are delayed until the respective slot arrives; this delay may be up to $T-1$ cycles. From the core's network interface to the memory, a latency of L_{down} cycles is added. Accessing the memory requires t_{slot} cycles. Transmitting the data back to the core adds L_{up} more cycles. The sum of these times determines the access latency observed at a core. The minimum TDM slot length depends however only on the maximum time the memory requires to serve a transaction.

Figure 3 shows that pipelining shifts the TDM slots in time along the path from the processor cores to the memory and back. Therefore, a core's slot starts at different times in different parts of that path. When the timing of the (SDRAM) memory controller is known (e.g., in [10]), the read and write commands can be sent from the client in the exact right point in time so they travel trough the network, arrive at the memory controller, and then arrive at the SDRAM bus without any flow control or buffering (except pipeline registers).

[1] Assuming equal bandwidth for all processing cores. Optimizing the bandwidth of the individual cores for the WCET has not been very beneficial [25].

For WCET analysis the timings for individual cache misses are: t_{wcrd} for a cache line fill and t_{wcwr} for a cache line write back. Compared to a round-robin arbitration, the constant time between two possible accesses can be used to tighten WCET bounds for a sequence of accesses. The time spent executing instructions that do not access memory (or are guaranteed hits) between two accesses can be subtracted from the TDM period T.

5 Implementation and Evaluation

We have implemented three different memory arbiters: (1) a round-robin arbiter, (2) a centralized TDM arbiter, and (3) the distributed TDM arbiter as described in Section 3. We used the high-level hardware description language Chisel [1], which allows the generation of Verilog code for logic synthesis. To evaluate the arbiter rather than the processor cores, we connected the arbiters to test driver cores (4, 8,. . . , 128), together with a PLL, and a memory controller. The design was synthesized and constrained to run at 200 MHz with Altera Quartus II. As target device we chose the Cyclone IV FPGA used on Altera's DE-115 development board (part number EP4CE115F29C7N).

All arbiters use the OCP communication interface used in T-CREST project. For a multicore with N cores, it has N OCP slave interfaces that receive data from the processor cores and one OCP master interface that sends data to the memory controller. To make the arbiter reusable, it is configurable with five parameters: number of cores, address width, data width, burst length, and controller delay. Additional parameters like slot length and period are derived from these parameters.

For a multicore with N cores, each core is given an *id*, $0, 1, \ldots N-1$. Based on the number of cores, a specific time slot is defined for each core in the TDM based arbiters. The slot length is defined as $t_{slot} = \max(t_{rd} + t_{ctrlrd}, t_{wr} + t_{ctrlwr})$, which is the amount of clock cycles for a read or write burst. The slot length and the number of cores is configurable. In a complete memory access period, $T = N \times t_{slot}$, each processing core has a chance for either a read or write burst. At each core a counter is used to generate the according single cycle enable signal.

For the centralized TDM arbiter, a finite state machine (FSM) controls the local TDM arbitration. When the *enable* signal is asserted, the FSM will transfer the memory access request of the corresponding node into the memory tree. The FSM has only three states: *idle*, *read*, and *write*.

For the distributed TDM arbiter, the arbitration is divided into N local arbiters. Each local arbiter has a counter for the TDM slot counting. The outputs of all local arbiters are fed to a tree of OR gates and into pipeline registers before reaching the memory controller. On the return path from the memory controller, this data is broadcasted to all the local arbiters. Two pipeline registers on the downstream path and one on the upstream path are enough to keep the critical path short enough for a 200 MHz operating frequency.[2]

Figure 4 shows the maximum frequency (Fmax) of each arbiter. The centralized arbiters (round-robin and central TDM) lead to slower clock frequency when increasing the number of processing cores. The simple round-robin arbiter can be clocked faster than the central TDM arbiter. In contrast, the maximum frequency of the distributed TDM arbiter remains relatively close to the input frequency (200 MHz), even as the number of nodes increases exponentially. The distributed TDM arbiter has the shortest critical path since the datapaths

[2] A RISC style processor core can be clocked at about 100 MHz in this FPGA family.

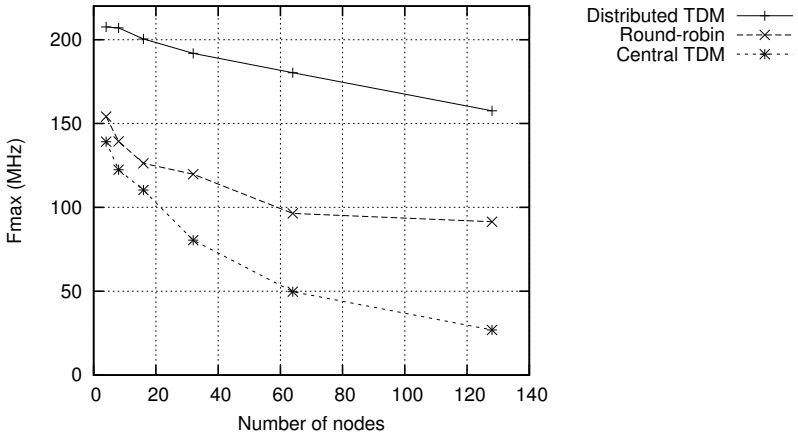

Figure 4 Maximum frequency (Fmax) for different arbiters and number of processor cores.

Table 2 Logic cell counts for different arbiters and different number of processor cores.

Number of processor cores	4	8	16	32	64	128
Round-robin	139	324	614	1267	2549	5114
Centralized TDM	235	446	969	1943	3893	7764
Distributed TDM	470	980	1894	3827	7575	10277

from each node to the memory, through the arbiter, are independent of each other and broken up with pipeline registers.

Table 2 shows the resource consumption of each arbiter in terms of logic cell (LC) count on the FPGA. To set the number in relation, a RISC style processor pipeline consumes about 2000–5000 LCs. The centralized TDM arbiter requires more logic cells than the round-robin arbiter.

The distributed TDM arbiter replicates some logic (e.g., the counters for the TDM slots) at the client side. Therefore, it consumes more resource, but also allows the highest clock frequency. Each client side component consumes about 30 LCs. However, the memory tree, made up of OR gates, consumes a large number of LCs. For every 4 cores, a 4-input look-up table (LUT) is needed for every bit of the OCP signals. As a rough estimate, about 55 logic cells are needed for the OR-gate tree per node. With a back-of-an-envelope calculation the LCs needed for the 128-core multicore can be estimated as: 30 LC per core * 128 + 55 LC per OR-gate tree * 128 = 10880. This confirms the synthesized results shown in Table 2.

Compared to centralized TDM arbitration, as used in [11] and [16], our pipelined design with the distributed arbitration at the individual nodes scales better with more nodes. The additional cost is moderate. The cost per node is in the range of 120 LCs, where a RISC style processor node itself consumes between 2000 and 5000 LCs.

6 Conclusion

Computer systems for real-time systems need to be time-predictable to allow static WCET analysis. For multicore processors with shared memory the access to this shared memory needs to be time-predictable as well. A time-division multiplexing (TDM) arbiter is time-predictable. It allows calculating the WCET of a task executing on one processor core

independent from other tasks executing on other cores. This paper presented a TDM memory network-on-chip with distributed arbitration and pipelining to achieve a high throughput through the interconnect. The additional latency through pipelining does not influence the slot length of the TDM schedule. The TDM schedule and the knowledge of the pipelined interconnect is the input for the WCET analysis of memory accesses for a multicore processor.

Source Access

The source of the described memory NoCs is open source under the simplified BSD licenses and available at GitHub within the Patmos project: `https://github.com/t-crest/patmos`. Details on the build process can be found in the Patmos reference handbook [20].

Acknowledgment. This work is part of the project "Hard Real-Time Embedded Multiprocessor Platform – RTEMP" and received partial funding from the Danish Research Council for Technology and Production Sciences under contract no. 12-127600. This work was partially funded under the European Union's 7th Framework Programme under grant agreement no. 288008: Time-predictable Multi-Core Architecture for Embedded Systems (T-CREST).

References

1 Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avizienis, John Wawrzynek, and Krste Asanovic. Chisel: constructing hardware in a scala embedded language. In Patrick Groeneveld, Donatella Sciuto, and Soha Hassoun, editors, *The 49th Annual Design Automation Conference (DAC 2012)*, pages 1216–1225, San Francisco, CA, USA, June 2012. ACM.

2 Aydin O Balkan, Gang Qu, and Uzi Vishkin. Mesh-of-trees and alternative interconnection networks for single-chip parallelism. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(10):1419–1432, 2009.

3 Christoph Cullmann, Christian Ferdinand, Gernot Gebhard, Daniel Grund, Claire Maiza, Jan Reineke, Benoît Triquet, and Reinhard Wilhelm. Predictability considerations in the design of multi-core embedded systems. In *Proceedings of Embedded Real Time Software and Systems*, May 2010.

4 Manil Dev Gomony, Benny Akesson, and Kees Goossens. Coupling tdm noc and dram controller for cost and performance optimization of real-time systems. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–6, March 2014.

5 Manel Djemal, François Pêcheux, Dumitru Potop-Butucaru, Robert De Simone, Franck Wajsburt, and Zhen Zhang. Programmable routers for efficient mapping of applications onto NoC-based MPSoCs. In *Design and Architectures for Signal and Image Processing (DASIP)*, pages 1–8, Oct 2012.

6 Jamie Garside and Neil C Audsley. Investigating shared memory tree prefetching within multimedia noc architectures. In *Memory Architecture and Organisation Workshop*, 2013.

7 Evangelia Kasapaki and Jens Sparsø. Argo: A Time-Elastic Time-Division-Multiplexed NoC using Asynchronous Routers. In *Proc. IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. IEEE Computer Society Press, 2014.

8 Timon Kelter, Heiko Falk, Peter Marwedel, Sudipta Chattopadhyay, and Abhik Roychoudhury. Static analysis of multi-core tdma resource arbitration delays. *Real-Time Systems*, 50(2):185–229, 2014.

9 Herman Kopetz. *Real-Time Systems*. Kluwer Academic, Boston, MA, USA, 1997.

10 Edgar Lakis and Martin Schoeberl. An SDRAM controller for real-time systems. In *Proceedings of the 9th Workshop on Software Technologies for Embedded and Ubiquitous Systems*, 2013.

11 Ben Lickly, Isaac Liu, Sungjun Kim, Hiren D. Patel, Stephen A. Edwards, and Edward A. Lee. Predictable programming on a precision timed architecture. In Erik R. Altman, editor, *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES 2008)*, pages 137–146, Atlanta, GA, USA, October 2008. ACM.

12 Thomas Lundqvist and Per Stenström. Timing anomalies in dynamically scheduled microprocessors. In *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS 1999)*, pages 12–21, Washington, DC, USA, 1999. IEEE Computer Society.

13 Jörg Mische, Stefan Metzlaff, and Theo Ungerer. Distributed memory on chip–bringing together low power and real-time. In *Proceedings of the Workshop on Reconciling Performance and Predictability (RePP)*, Grenoble, France, 2014.

14 OCP-IP Association. Open core protocol specification 2.1. http://www.ocpip.org/, 2005.

15 Christof Pitter and Martin Schoeberl. Time predictable CPU and DMA shared memory access. In *International Conference on Field-Programmable Logic and its Applications (FPL 2007)*, pages 317–322, Amsterdam, Netherlands, August 2007. IEEE.

16 Christof Pitter and Martin Schoeberl. A real-time Java chip-multiprocessor. *ACM Trans. Embed. Comput. Syst.*, 10(1):9:1–34, 2010.

17 Abbas Rahimi, Igor Loi, Mohammad Reza Kakoee, and Luca Benini. A fully-synthesizable single-cycle interconnection network for shared-L1 processor clusters. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pages 1–6, March 2011.

18 Jakob Rosen, Alexandru Andrei, Petru Eles, and Zebo Peng. Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip. In *Proceedings of the Real-Time Systems Symposium (RTSS 2007)*, pages 49–60, Dec. 2007.

19 Martin Schoeberl. Time-predictable computer architecture. *EURASIP Journal on Embedded Systems*, vol. 2009, Article ID 758480:17 pages, 2009.

20 Martin Schoeberl, Florian Brandner, Stefan Hepp, Wolfgang Puffitsch, and Daniel Prokesch. *Patmos Reference Handbook*. Technical University of Denmark, 2014.

21 Martin Schoeberl, Florian Brandner, Jens Sparsø, and Evangelia Kasapaki. A statically scheduled time-division-multiplexed network-on-chip for real-time systems. In *Proceedings of the 6th International Symposium on Networks-on-Chip (NOCS)*, pages 152–160, Lyngby, Denmark, May 2012. IEEE.

22 Martin Schoeberl, Pascal Schleuniger, Wolfgang Puffitsch, Florian Brandner, Christian W. Probst, Sven Karlsson, and Tommy Thorn. Towards a time-predictable dual-issue microprocessor: The Patmos approach. In *First Workshop on Bringing Theory to Practice: Predictability and Performance in Embedded Systems (PPES 2011)*, pages 11–20, Grenoble, France, March 2011.

23 Andreas Schranzhofer, Jian-Jia Chen, and Lothar Thiele. Timing analysis for TDMA arbitration in resource sharing systems. In Marco Caccamo, editor, *16th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2010, Stockholm, Sweden, April 12-15, 2010*, pages 215–224. IEEE Computer Society, 2010.

24 Jens Sparsø, Evangelia Kasapaki, and Martin Schoeberl. An area-efficient network interface for a TDM-based network-on-chip. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '13, pages 1044–1047, San Jose, CA, USA, 2013. EDA Consortium.

25 Jack Whitham and Martin Schoeberl. The limits of TDMA based memory access scheduling. Technical Report YCS-2011-470, University of York, 2011.